Chapter 3

Mano's Basic Computer

REVIEW

- **We introduced registers**
- **We discussed the kinds of Transfer, Arithmetic, Logic and Shift Microoperations**
- **We discussed general aspects of CPU, Memory and Bus architectures**

- **In this chapter we introduce a more complicated <u>basic computer and show how its operation can be specified with register transfer statements</u>**

CONSIDERING THE NEXT PROBLEM IN DESIGN

A computer organization involves combining everything we have learned to date into a single integrated unit

- What is a computer?

Von Neuman refers to a computer as a "stored program digital computer"

- What is a program?
  - What is an instruction?
  - How are instructions executed?

GOALS

We conclude our lecture series by considering computer organization

- Combining the CPU, Memory and Bus architectures

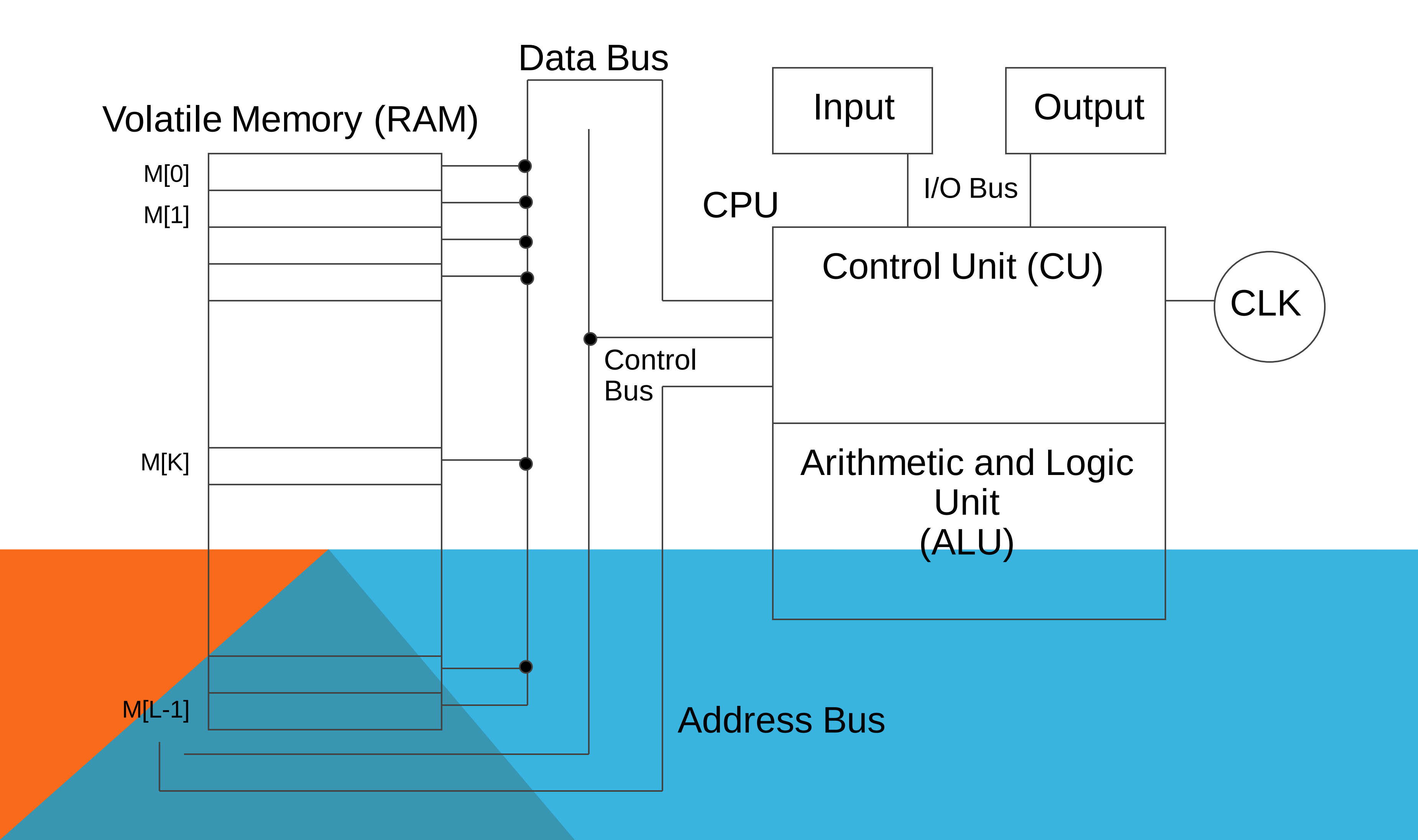
We introduce the concept of an instruction

- Instruction design and architecture
- Microoperation sequencing (timing, control)
- Roles of different registers

We will follow a model of a virtual/logical computer organization adapted from M. Mano (Computer System Architecture, 3d Edition)

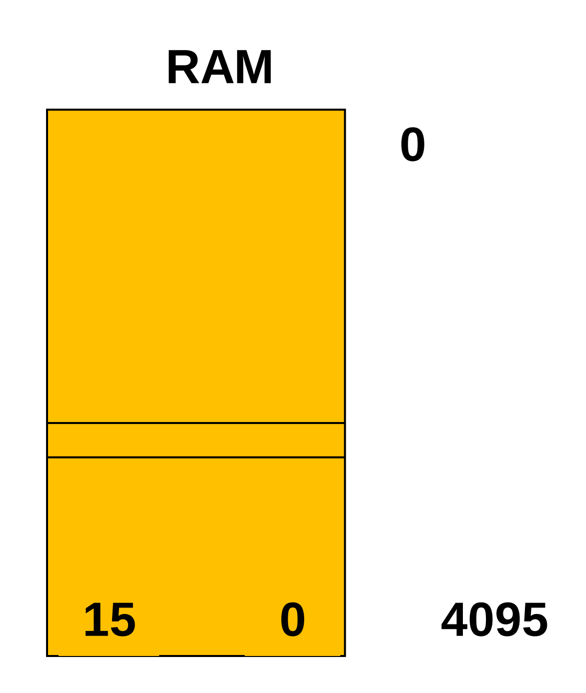
COMPUTER – HIGH LEVEL VIEW

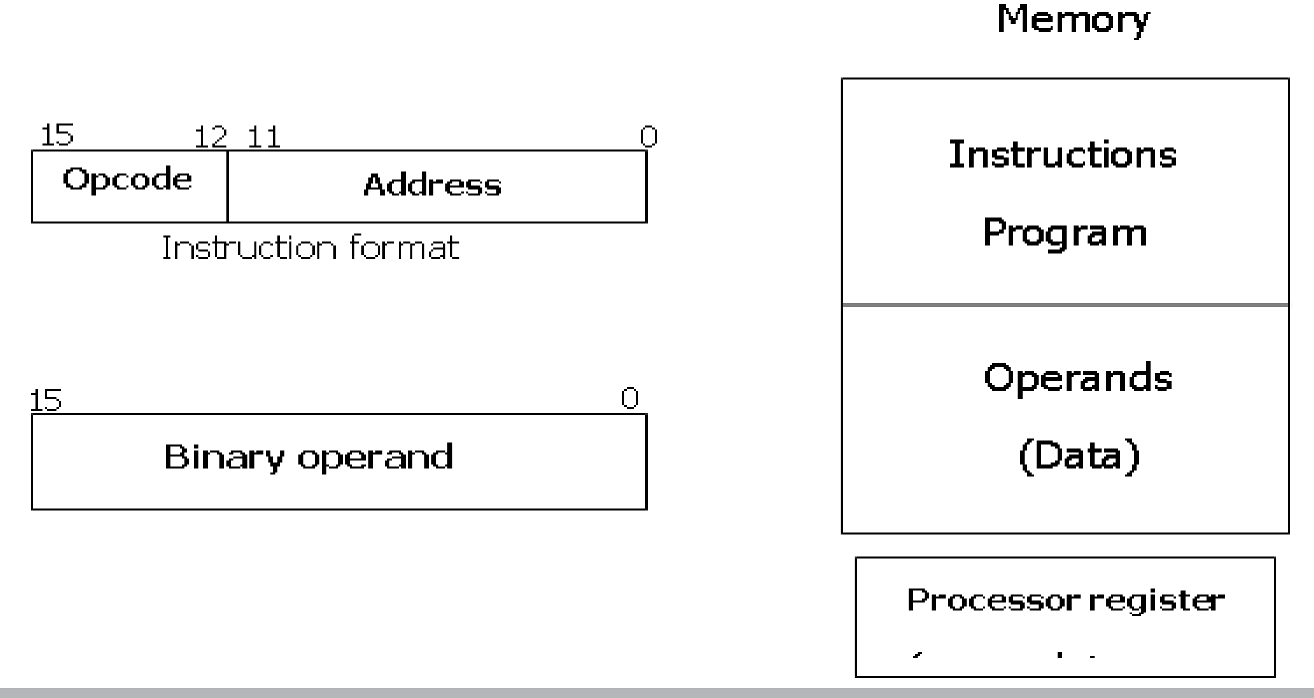
CPU, Memory and Bus architectures with interface to I/O.



MANO'S COMPUTER SPECIFICATION

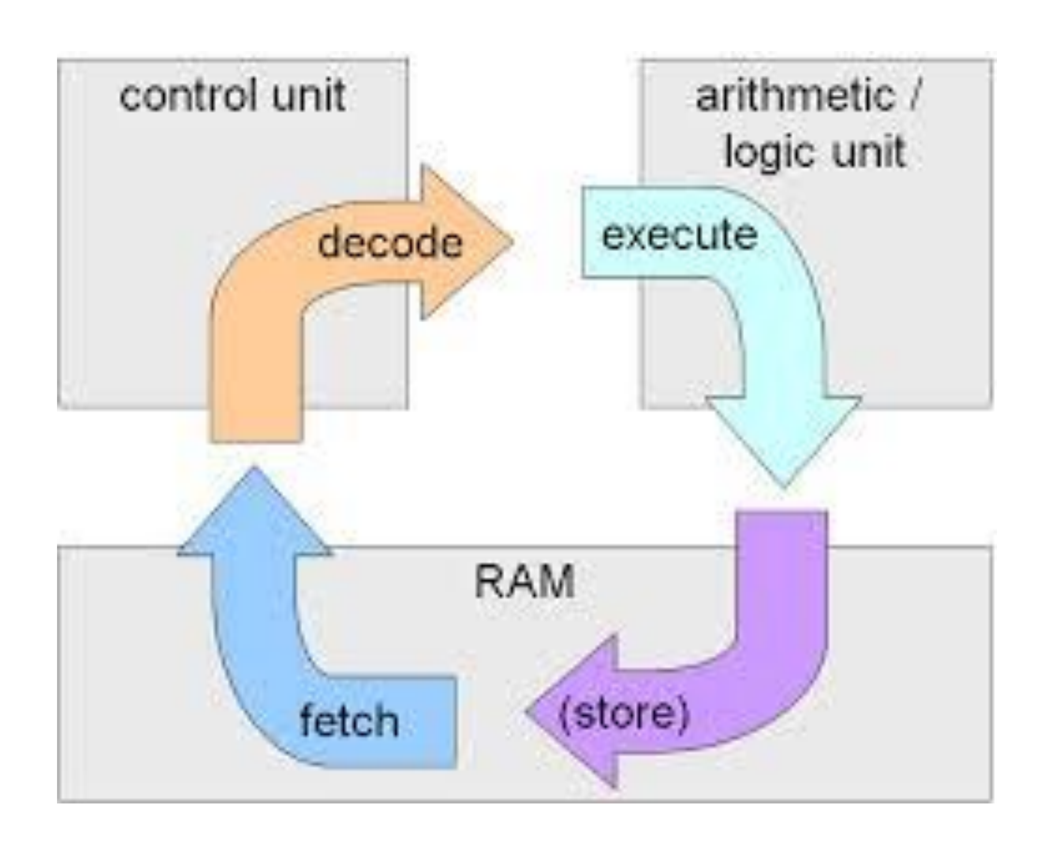
- Mano has a Memory (RAM) of 4096 words.
- This requires an Address bus of 12 bits.
- Each word is 16 bits long.
- The Data bus carries exactly 1 word of data between Memory and CPU.
- It has 8 registers.
- Input and Output is defined using the ASCII code of length 8 bits.





- Instruction is stored in one 16-bit memory word,
- It consists 4 bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand (data used).
- The control reads a 16-bit instruction from the program portion of memory. It then executes the operation specified by the operation code.
- Computers that have a single-processor register usually assign to it the name Accumulator and label it AC.

FETCH – DECODE – EXECUTE CYCLE



THE MANO MODEL OF THE CPU REGISTERS

CPU registers used in the model (Mano), categorized by <u>length</u>:

- PC :: Program counter (hold address of next inst. – 12 bits)
- AR :: Address register (holds address for memory– 12 bits)

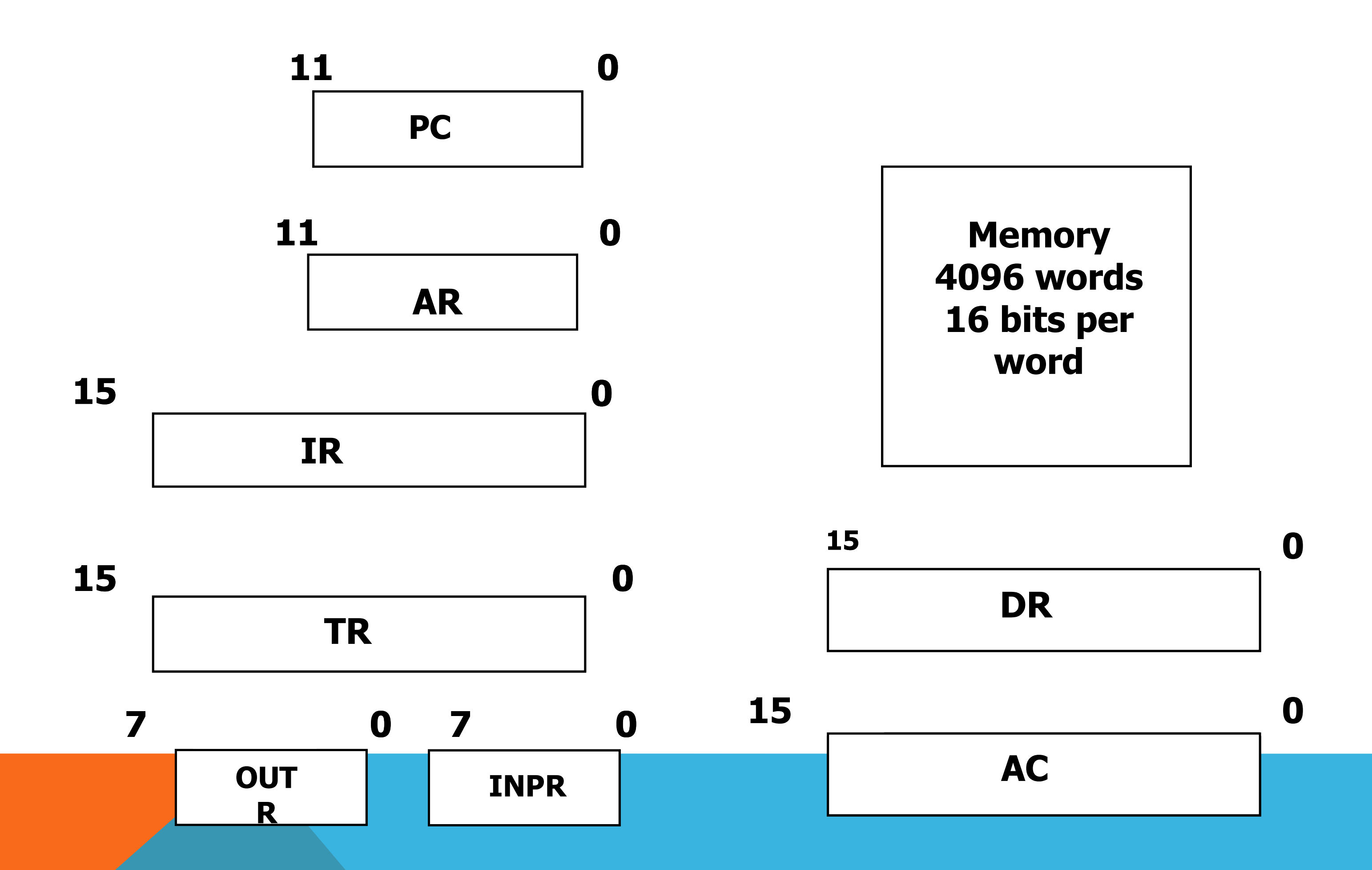
- DR :: Data register (holds memory operand– 16 bits)
- AC :: Accumulator (processor register holds data – 16 bits)
- IR :: Instruction register (holds instruction code – 16 bits)
- TR :: Temporary register (holds temporary data – 16 bits)

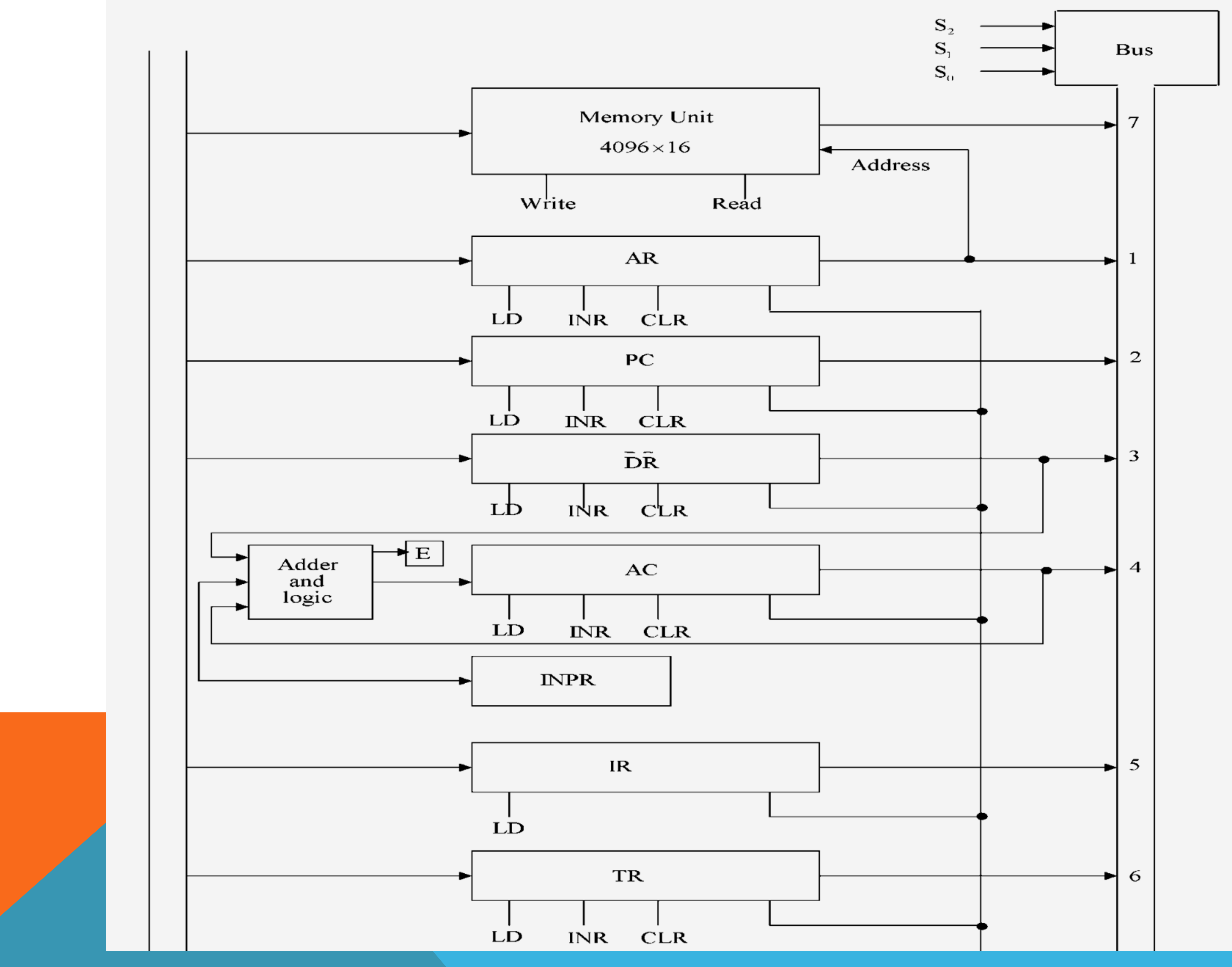
- INR :: Input buffer register (holds input ASCII data – 8 bits)
- OUTR :: Output buffer register (holds output ASCII data – 8 bits)

SCR :: Sequence counter register (4 bits)

E, R :: Single bit flip-flops (flag/utility, interrupt)

BASIC COMPUTER REGISTERS AND MEMORY





S_2
S_1
S_0
Bus
Memory Unit
4096 × 16
7
Address
Write
Read
AR
1
LD
INR
CLR
PC
2
LD
INR
CLR
DR
3
LD
INR
CLR
Adder
and
logic
E
AC
4
LD
INR
CLR
INPR
IR
5
LD
TR
6
LD
INR
CLR

STORED PROGRAMS

- A stored program is a set of instructions and data expressed in binary language, stored in non-volatile (ie. disk storage) memory

Programs can be executed only from Memory.

- Thus, a program must be loaded from disk to RAM in order to execute it.
- Loaded programs are called *processes*.
- Individual *instructions* must be transferred to the CPU where they are executed, using data that must be obtained from either CPU registers or RAM

- A process is executed by executing each individual instruction that, collectively, fulfill the intentions of the programmer.

COMPUTER INSTRUCTION

- **A computer instruction is a binary code that specifies a sequence of microoperations for the computer.**
- **Instruction codes together with data are stored in memory (RAM).**
- **The computer reads each instruction from memory and places it in a control register.**
- **The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations.**
- **Operations such as: add, subtract, multiply, shift, and complement.**
- The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.

RELATIONSHIP BETWEEN A COMPUTER OPERATION AND A MICROOPERATION

- **An operation is <u>part of an instruction stored in computer memory</u>. It is a binary code that tells the computer to perform a specific operation.**
- **For every operation code, the control issues a sequence of microoperations needed for the hardware implementation of the specified operation.**
- **An instruction code must specify not only the operation but also the registers (by binary code) or the memory words (by address) where the operands are to be found, as well as the register or memory word where the result is to be stored.**

INSTRUCTION ARCHITECTURE

The list of all instructions engineered for a computer's CPU is called the *instruction set*.

To distinguish each instruction, they are assigned a unique numeric code

- Can be done in many ways
- Does not need to be ordinal (ie. starting from 0 and running contiguously)
- Can be partially ordinal and partially hierarchical
  - Mano's approach

Instructions must be capable of referencing Memory and/or CPU addresses in order to cause data to be transferred and operated on.

INSTRUCTION FORMAT

Instructions usually consist of:

- Operation code (opcode) part
- Address part
- Addressing Mode part
  - Direct addressing – data found at the specified address
  - Indirect addressing – data found at the address found at the specified address (pointer concept)

The number of memory referenced operands varies from computer to computer.

Mano simplifies the instruction architecture to permit zero or one memory address field. In the case of Indirect addressing mode there are two memory accesses, but only one reference.

0

1

15 14 12 11 0

IR

I	OpCode	Address

Mode bit

MANO’S INSTRUCTION FORMAT

- **In Mano’s Computer, since the memory contains 4096 (= 2^{12}) words, we needs 12 bit to specify which memory address this instruction will use**
- **Bit 15 of the instruction specifies the *addressing mode* (0: direct addressing, 1: indirect addressing)**
- **3 bits for the instruction’s opcode (possible 8 operations)**

MANO’S COMPUTER INSTRUCTION FORMAT

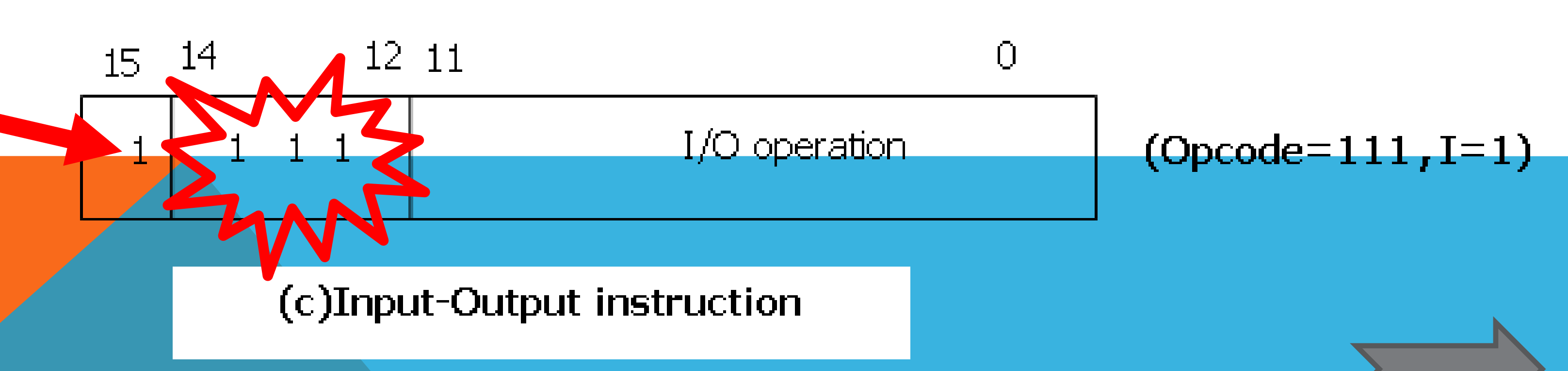
14	12 11	0
I	opcode	Address

(a)Memory –reference

15	14 12	11 0
0	1 1 1	Register operation

(Opcode=111,I=0)

(b)Register –reference instruction



(Opcode=111,I=1)

(c)Input-Output instruction

INSTRUCTION HIERARCHY

Mano's instruction set consists of 25 instructions:

Instruction

- OpCode (000-110)
  - Direct (I=0)
  - Indirect (I=1)
- OpCode (111)
  - Direct (I=0)
  - Indirect (I=1)

4 bit code		4 bit code		16 bit code		16 bit code	
0	AND	8	AND	7800	CLA	F800	INP
1	ADD	9	ADD	7400	CLE	F400	OUT
2	LDA	A	LDA	7200	CMA	F200	SKI
3	STA	B	STA	7100	CME	F100	SKO
4	BUN	C	BUN	7080	CIR	F080	ION
5	BSA	D	BSA	7040	CIL	F040	IOF
6	ISZ	E	ISZ	7020	INC		
				7010	SPA		
				7008	SNA		
				7004	SZA		
				7002	SZE		
				7001	HLT		

7

12

6

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

INSTRUCTION SET COMPLETENESS

Selection of instructions should span a variety of applications suitable to support programming

- Arithmetic, logical and shift instructions
- Instructions for moving data to and from memory and CPU registers
- Program control instructions, instructions that check status conditions
- Input and Output instructions

INSTRUCTION SET COMPLETENESS

Set of instructions using which user can construct machine language programs to evaluate any computable function.

- Instruction Types

  Functional Instructions
  - Arithmetic, logic, and shift instructions
  - ADD, CMA, INC, CIR, CIL, AND, CLA (other than ADD/AND?)

  Transfer Instructions
  - Data transfers between the main memory and the processor registers
  - LDA, STA

  Control Instructions
  - Program sequencing and control
  - BUN, BSA, ISZ

  Input/Output Instructions
  - Input and output
  - INP, OUT

INSTRUCTION CYCLE

- **In Computer, a machine instruction is executed in the following cycle:**
  1. Fetch an instruction from memory
  2. Decode the instruction and calculate effective address (EA)
  3. Read the EA from memory if the instruction has an indirect address (Fetch operand)
  4. Execute the instruction

- **Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction.**
- **This process continues indefinitely unless a HALT instruction is encountered.**

FETCH AND DECODE CYCLE

Microoperations for the fetch and decode phases can be specified by the following register transfer statements:

$T_{0:}$ AR←PC

$T_{1:}$ IR←M[AR] ; PC←PC+1

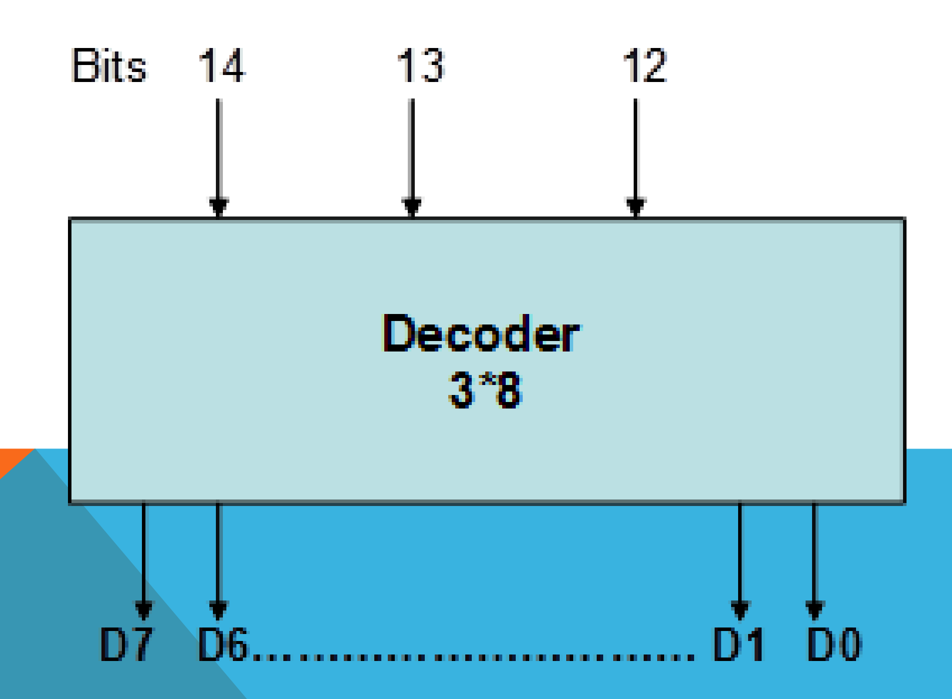
T_3 : D_0 ,…,D_7 Decode IR(12-14) ,AR←IR(0-11) ,I←IR(15)

Fetch from memory the current instruction (which address is in PC)

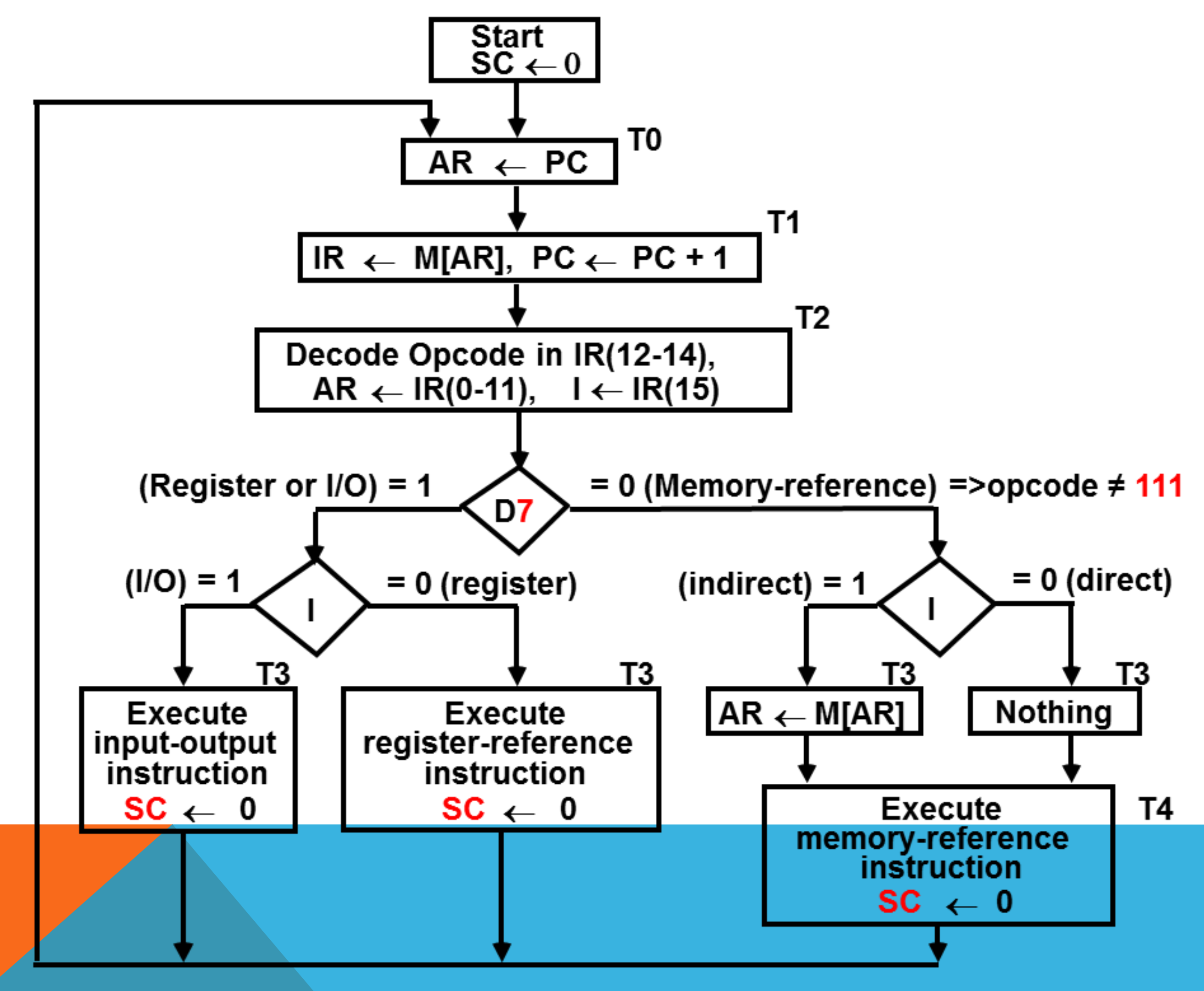
Decode instruction found now in IR

DETERMINE THE TYPE OF INSTRUCTION

- We use bits 12 , 13, 14 as input to the decoder to determine the type of instruction
- Only one of the decoder outputs = 1 at certain combination (i.e. operation) of the inputs.

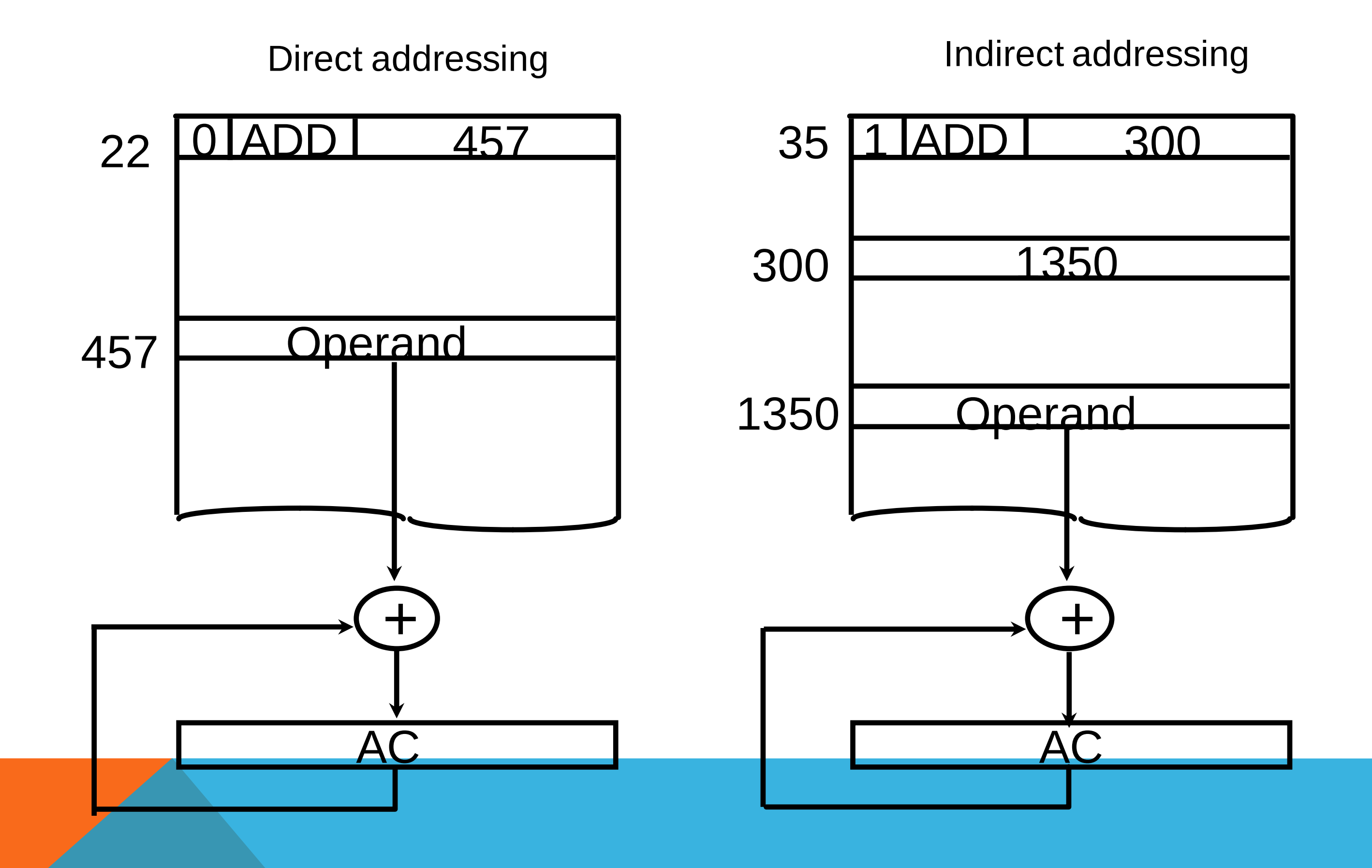


DETERMINE THE TYPE OF INSTRUCTION



D'_7IT_3:	AR ← M[AR]
$D'_7I'T_3$:	Nothing
$D_7I'T_3$:	Execute a register-reference instr.
D_7IT_3:	Execute an input-output instr.

DIFFERENCE BETWEEN DIRECT AND INDIRECT ADDRESSING



1. MEMORY-REFERENCE INSTRUCTIONS:

Symbol	Operation Decoder	Symbolic description
AND	D_0	AC← AC ^M[AR]
ADD	D_1	AC←AC + M[AR] , E← C_{OUT}
LDA (load to accumulator)	D_2	AC← M[AR]
STA (Store from accumulator)	D_3	M[AR] ← AC
BUN (branch unconsitionally)	D_4	PC ← AR
BSA	D_5	M[AR] ← PC , PC←AR+1
ISZ	D_6	M[AR] ← M[AR]+1, IF M[AR]+1=0 then

D0: AND: AND TO AC

- This is an instruction that performs the AND logic operation on pairs of bits in *AC* and the memory word specified by the effective address found in (AR) now.
- We can not work on the memory location directly, we have to bring the data to DR (Data Register) first.
- The result of the operation is transferred to *AC*.
- The microoperations that execute this instruction are:

D_0T_4: DR← M[AR]

D_0T_5: AC← AC ^DR, SC← 0

D1: ADD: ADD TO AC

- **This instruction adds the content of the memory word specified by the effective address found in (AR) now to the value of *AC*.**
- **We can not work on the memory location directly, we have to bring the data to DR (Data Register) first.**
- **The result of the operation is transferred to *AC*.**
- **The microoperations that execute this instruction are:**

D_1T_4: DR← M[AR]

D_1T_5: AC← AC +DR, E←Cout, SC← 0

D2: LDA: LOAD TO AC

- **This instruction transfers the memory word specified by the effective address found in (AR) now to AC.**
- **We can not work on the memory location directly, we have to bring the data to DR (Data Register) first.**
- **The microoperations that execute this instruction are:**

D_2T_4: DR← M[AR]

D_2T_5: AC← DR, SC← 0

D3: STA: STORE AC

- **This instruction stores the content of AC into the memory word specified by the effective address found in (AR) now.**
- **The microoperation that execute this instruction is:**

D_3T_4: $M[AR] \leftarrow AC$, $SC \leftarrow 0$

D4: BUN: BRANCH UNCONDITIONALLY

- **This instruction transfers the program to the instruction specified by the effective address found in (AR) now.**
- **The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally.**
- **The effective address from AR is transferred through the common bus to PC.**
- **The microoperation that execute this instruction is:**

$$D_4T_4: \quad PC \leftarrow AR \quad , SC \leftarrow 0$$

D5: BSA: BRANCH AND SAVE RETURN ADDRESS

- **This instruction is useful for branching to a portion of the program called a subroutine or procedure.**
- **When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address found in (AR) now.**
- **The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine.**
- **The microoperation that execute this instruction is:**

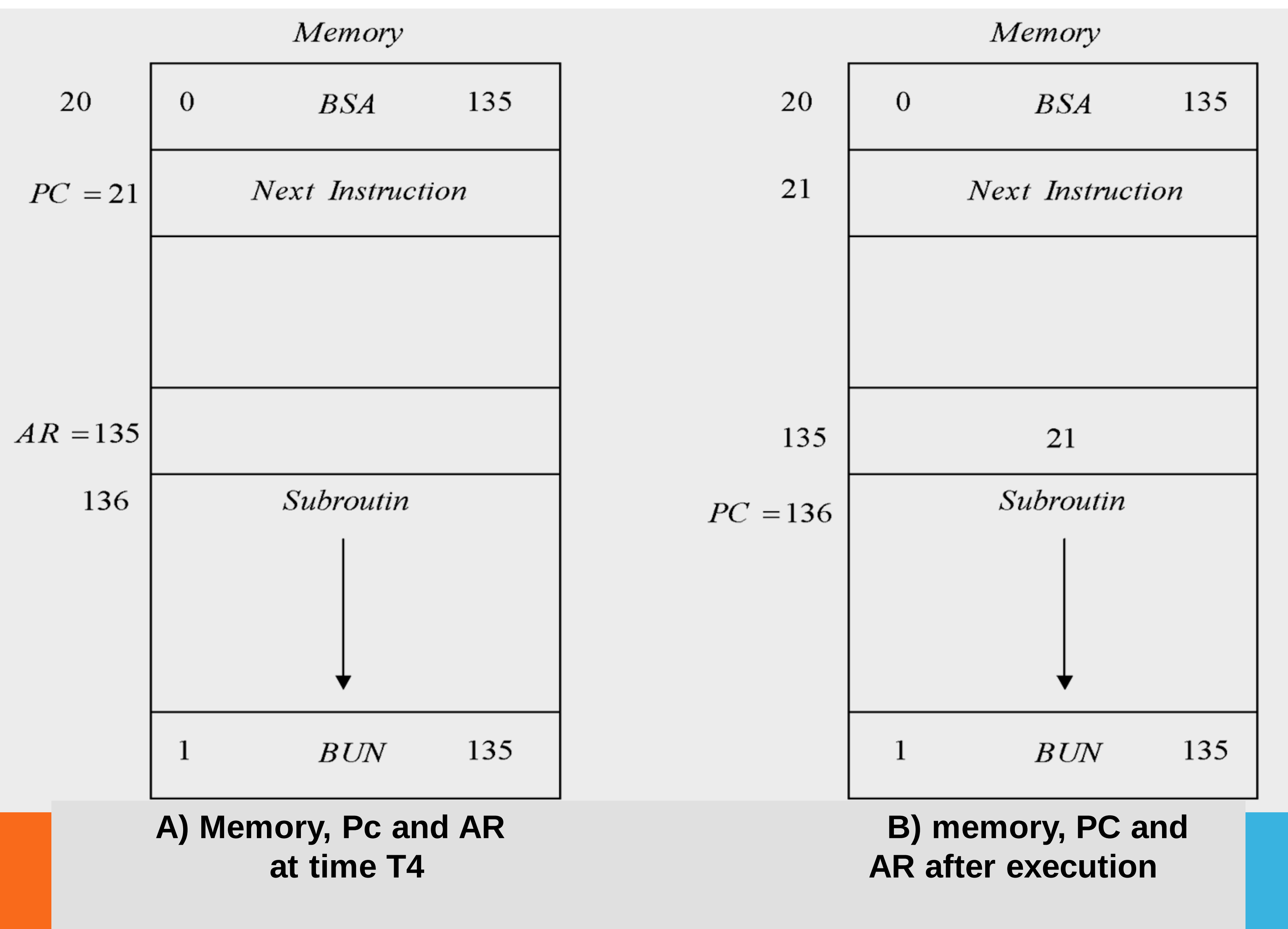
D_4T_4: $M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$,

D_5T_4: $PC \leftarrow AR$, $SC \leftarrow 0$

EXAMPLE ON BSA

- **The BSA instruction is assumed to be in memory at address 20**
- **The *I bit* is 0 and the address part of the instruction has the binary equivalent of 135.**
- **After the fetch and decode phases, PC contains 21, which is the address of the next instruction in the program (referred to as the return address).**
- **AR holds the effective address 135.**
- **The BSA instruction performs the following numerical operation:**

$$M[135] \leftarrow 21\ ,\ PC \leftarrow 135 + 1 = 136$$



A) Memory, Pc and AR at time T4

B) memory, PC and AR after execution

Figure 3.8 Example of BSA instruction execution.

D6: ISZ: INCREMENT AND SKIP IF ZERO

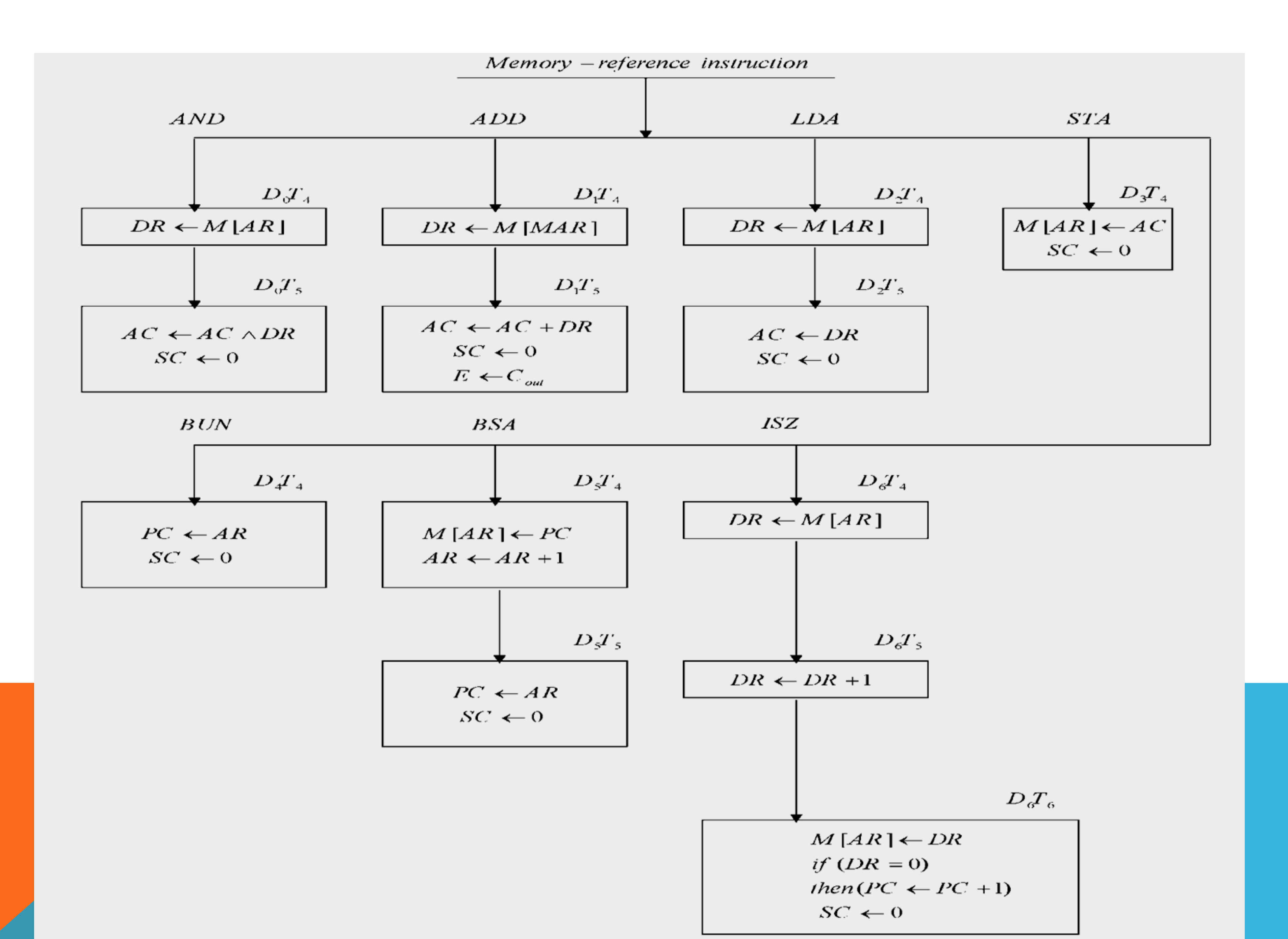
- **This instruction increments the word specified by the effective address found in (AR) now, and if the incremented value is equal to 0, PC is incremented by 1.**
- **The programmer usually stores a negative number (in 2's complement) in the memory word (AR).**
- **As this negative number is repeatedly incremented by one, it eventually reaches the value of zero.**
- **At that time PC is incremented by one in order to skip the next instruction in the program.**
- **Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory.**

- The microoperations that execute this instruction are:

D_6T_4: $DR \leftarrow M[AR]$

D_6T_5: $DR \leftarrow DR+1$

D_6T_6: $M[AR] \leftarrow DR$, if($DR = 0$) then ($PC \leftarrow PC + 1$), $SC \leftarrow 0$



Memory – reference instruction
AND
ADD
LDA
STA
D_0T_4
$DR \leftarrow M[AR]$
D_0T_5
$AC \leftarrow AC \wedge DR$
$SC \leftarrow 0$
D_1T_4
$DR \leftarrow M[MAR]$
D_1T_5
$AC \leftarrow AC + DR$
$SC \leftarrow 0$
$E \leftarrow C_{out}$
D_2T_4
$DR \leftarrow M[AR]$
D_2T_5
$AC \leftarrow DR$
$SC \leftarrow 0$
D_3T_4
$M[AR] \leftarrow AC$
$SC \leftarrow 0$
BUN
BSA
ISZ
D_4T_4
$PC \leftarrow AR$
$SC \leftarrow 0$
D_5T_4
$M[AR] \leftarrow PC$
$AR \leftarrow AR + 1$
D_5T_5
$PC \leftarrow AR$
$SC \leftarrow 0$
D_6T_4
$DR \leftarrow M[AR]$
D_6T_5
$DR \leftarrow DR + 1$
D_6T_6
$M[AR] \leftarrow DR$
$if\ (DR = 0)$
$then (PC \leftarrow PC + 1)$
$SC \leftarrow 0$